//lis

int A[] = {1,6,2,3,7,5};

int d[6], g[10], b[10];

const int INF = 10000;

int main()

{

int n = 6;

for (int i = 1; i <= n; i++) g[i] = INF;

for (int i = 0; i < n; i++) {

int k = lower\_bound(g+1, g+n+1, A[i]) - g;

d[i] = k;

g[k] = A[i];

}

for (int i = 0; i < n; i++)

printf("%d %d\n", d[i], b[i]);

}

//lcs

//start from index one

int s1[N], s2[N], s1\_len, s2\_len, dp[N][N], pre[N][N];

void LCS()

{

memset(dp, 0, sizeof(dp));

for (int i = 1; i <= s1\_len; i++)

for (int j = 1; j <= s2\_len; j++)

if (s1[i] == s2[j]) {

dp[i][j] = dp[i-1][j-1] + 1;

pre[i][j] = LEFT\_UP;

} else {

if (dp[i-1][j] < dp[i][j-1]) {

dp[i][j] = dp[i][j-1];

pre[i][j] = LEFT;

} else {

dp[i][j] = dp[i-1][j];

pre[i][j] = UP;

}

}

}

void print\_LCS(int i, int j)

{

if (i == 0 || j == 0) return;

if (pre[i][j] == LEFT\_UP) {

print\_LCS(i-1, j-1);

cout << s1[i];

}

else if (pre[i][j] == UP)

print\_LCS(i-1, j);

else if (pre[i][j] == LEFT)

print\_LCS(i, j-1);

}

//ap

vector<int> adj[N];

int clk[N], low[N];

int ans, t;

// dfs(1, -1)

void dfs(int cur, int parent)

{

int child = 0;

bool flag = false;

low[cur] = clk[cur] = t;

t++;

for (int i = 0; i < adj[cur].size(); i++) {

int next = adj[cur][i];

if (!clk[next]) {

child++;

dfs(next, cur);

low[cur] = min(low[cur], low[next]);

if (low[next] >= clk[cur])

flag = true;

} else if (next != parent) {

low[cur] = min(low[cur], clk[next]);

}

}

if (parent == -1 && child >= 2)

ans++;

else if (parent != -1 && flag)

ans++;

}

/\* 二分圖匹配

\* v 是 adj list

\* vis每次要memset

\* match[y] == 0 表示還沒配對到

\*/

bool bipartite(int x)

{

for (int i = 0; i < v[x].size(); i++) {

int y = v[x][i];

if (vis[y] == 0) {

vis[y] = 1;

if (match[y] == 0 || bipartite(match[y])) {

match[y] = x;

return true;

}

}

}

return false;

}

//euler path

int adj[N][N];

int degree[N];

deque< pair<int,int> > edge;

bool evenDegree()

{

memset(degree, 0, sizeof(degree));

for (int x = 0; x < N; x++)

for (int y = x; y < N; y++) {

degree[x] += adj[x][y];

degree[y] += adj[x][y];

}

for (int x = 0; x < N; x++)

if (degree[x]%2)

return false;

return true;

}

void findEulerCircuit(int x)

{

for (int y = 0; y < N; y++)

if (adj[x][y] > 0) {

adj[x][y]--;

if (x != y) adj[y][x]--;

findEulerCircuit(y);

edge.push\_front( make\_pair(x,y) );

}

}

//euler path

int main()

{

int a, b;

while (cin >> a >> b) {

adj[a][b]++;

if (a != b)

adj[b][a]++;

}

findEulerCircuit(0);

deque< pair<int,int> > :: iterator it;

for (it = edge.begin(); it != edge.end(); it++)

cout << it->first << ' ' << it->second << endl;

return 0;

}

//max flow

// adj matrix

// 1 is start

// m is end

long long adj[210][210];

int pre[210];

long long vst[210];

int n, m;

bool find\_path(long long x)

{

queue<int> q;

q.push(1);

bool ok = false;

while (!q.empty()) {

int cur = q.front(); q.pop();

if (cur == m) {

ok = true;

break;

}

for (int i = 0; i < 210; i++) {

if (adj[cur][i] && vst[i] < x) {

vst[i] = x;

q.push(i);

pre[i] = cur;

}

}

}

return ok;

}

//max flow

long long find\_min()

{

int cur = pre[m];

long long ret = adj[pre[m]][m];

while (cur != 1) {

ret = min(ret, adj[pre[cur]][cur]);

cur = pre[cur];

}

return ret;

}

void flow(long long x)

{

int cur = m;

while (cur != 1) {

adj[pre[cur]][cur] -= x;

adj[cur][pre[cur]] += x;

cur = pre[cur];

}

}

long long ans = 0;

long long t = 1;

while (find\_path(t++)) {

long long x = find\_min();

flow(x);

ans += x;

}

cout << ans << endl;

//scc

vector <int> adj[N]; //adj list

vector <int> rev[N]; //adj list reverse

vector <int> finish; //the order of leaving 1st dfs

vector <int> scc[N];

bool vst[N];

void dfs1(int x) //first dfs to get the order of leaving dfs

{

vst[x] = true;

for (int i = 0; i < adj[x].size(); i++)

if (!vst[adj[x][i]])

dfs1(adj[x][i]);

finish.push\_back(x);

}

void dfs2(int x, int c) //use the reverse adj list to dfs

{

scc[c].push\_back(x);

vst[x] = true;

for (int i = 0; i < rev[x].size(); i++)

if (!vst[rev[x][i]])

dfs2(rev[x][i], c);

}

//scc

int main()

{

int a, b, n;

cin >> n;

while (cin >> a >> b) {

adj[a].push\_back(b);

rev[b].push\_back(a);

} //build graph

memset(vst, false, sizeof(vst));

for (int i = 0; i < n; i++)// first dfs

if (!vst[i])

dfs1(i);

int c = 0;

memset(vst, false, sizeof(vst));

for (int i = finish.size()-1; i >= 0; i--) //second dfs using the order of finish from the end

if (!vst[finish[i]])

dfs2(finish[i], c++);

for (int i = 0; i < c; i++) {

cout << "group " << i << "\n\t";

for (int j = 0; j < scc[i].size(); j++)

cout << scc[i][j];

cout << endl;

}

return 0;

}

//topo

vector <int> adj[N]; //adj list

int degree[N]; //in degree

bool vst[N];

int main()

{

int a, b, n;

cin >> n;

memset(degree, 0, sizeof(degree));

memset(vst, false, sizeof(vst));

while (cin >> a >> b) {

adj[a].push\_back(b);

degree[b]++;

} //build graph

queue <int> q;

for (int i = 0; i < n; i++)

if (degree[i] == 0)

q.push(i); //push degree 0 in queue

for (int i = 0; i < n; i++) {

int s = q.front(); q.pop();

vst[s] = true;

cout << s << ' ';

for (int j = 0; j < v[s].size(); j++)//vector s can reach

if (!vst[v[s][j]]) {

degree[v[s][j]]--;

if (degree[v[s][j]] == 0)

q.push(v[s][j]);

}

}

cout << endl;

return 0;

}

//mp string

//a long, b short

int fail[N];

char a[N], b[N];

vector<int> match;

int lena, lenb;

void make\_failure()

{

int n = lenb;

fail[0] = -1;

fail[1] = 0;

for (int i = 2; i <= n; i++) {

int tmp = fail[i-1];

while (1) {

if (b[tmp] == b[i-1]) {

fail[i] = tmp+1;

break;

} else if (tmp == 0) {

fail[i] = 0;

break;

}

tmp = fail[tmp];

}

}

}

//mp string

void str\_match()

{

match.clear();

int i = 0;

int j = 0;

int cnt = 0;

while (i < lena) {

while (a[i] == b[j] && i < lena) {

i++;

j++;

if (j == lenb) {

match.push\_back(i);

j = fail[j];

}

}

j = fail[j];

if (j == -1) {

i++;

j = 0;

}

}

}